Accelerating Model Reduction of Large Linear Systems with Graphics Processors

P. Benner[1], P. Ezzatti[2], D. Kressner[3], E. S. Quintana-Ortí[4], and A. Remón[4]

[1] Max-Planck-Institute for Dynamics of Complex Technical Systems (Magdeburg, Germany)
benner@mpi-magdeburg.mpg.de
[2] Centro de Cálculo-Instituto de la Computación, Universidad de la República (Montevideo, Uruguay)
pezzatti@fing.edu.uy
[3] Seminar für Angewandte Mathematik, ETHZ (Zürich, Switzerland)
daniel.kressner@sam.math.ethz.ch
[4] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I (Castellón, Spain)
{quintana,remon}@icc.uji.es

Abstract Model order reduction of dynamical linear time-invariant system appears in many applications from science and engineering. Numerically reliable SVD-based methods for this task require in general $\mathcal{O}(n^3)$ floating-point arithmetic operations, with n being in the range $10^3 - 10^5$ for many practical applications. In this paper we investigate the use of graphics processors (GPUs) to accelerate model reduction of large-scale linear systems by off-loading the computationally intensive tasks to this device. Experiments on a hybrid platform consisting of state-of-the-art general-purpose multi-core processors and a GPU illustrate the potential of this approach.

Key words: model reduction, dynamical linear systems, Lyapunov equations, SVD-based methods, GPUs.

1 Introduction

Model order reduction is an important numerical tool to diminish the simulation time or the cost of designing optimal controllers in many industrial processes, with dynamics modeled by a linear time-invariant (LTI) system:

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1}$$

Here, $x(t)$ contains the states of the system, with initial state $x^0 \in \mathbb{R}^n$, $u(t)$ and $y(t)$ contain the inputs and outputs, respectively, and $E, A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$. The system in (1) can also be described by the associated transfer function matrix (TFM) $G(s) = C(sE - A)^{-1}B + D$. A particularly important property is that the number of states (also known as the state-space

dimension or the order) of the system, n, is in general much larger than the number of inputs and outputs, m and p, respectively..

The goal of model reduction is to find a reduced-order LTI system,

$$\begin{aligned} \hat{E}\dot{\hat{x}}(t) &= \hat{A}\hat{x}(t) + \hat{B}u(t), \quad t > 0, \quad \hat{x}(0) = \hat{x}^0, \\ \hat{y}(t) &= \hat{C}\hat{x}(t) + \hat{D}u(t), \end{aligned} \tag{2}$$

of order r, with $r \ll n$, and associated TFM $\hat{G}(s) = \hat{C}(sE - \hat{A})^{-1}\hat{B} + \hat{D}$ which approximates the dynamics of the original system defined by $G(s)$. The reduced-order realization (2) can then replace the original high-order system in a simulation or the design of an optimal controller, thus simplifying such tasks considerably. Model reduction of large-scale systems appears, e.g., in thermal, thermo-mechanical, electro-mechanical and acoustic finite element models [1]. We consider a system to be large-scale if $n \sim \mathcal{O}(1,000) - \mathcal{O}(100,000)$; while, often, $m, p \sim \mathcal{O}(10) - \mathcal{O}(100)$.

The numerical method for model reduction considered in this paper is based on the so-called state-space truncation approach and requires, at an initial stage, the solution of two coupled generalized Lyapunov equations. The reduced-order system is then obtained using a variant of the balanced truncation (BT) method [2,3], which only requires a few dense linear algebra computations. Although there exist several other approaches for model reduction (see, e.g., [1,4] and the references therein), those are specialized for certain problem classes and often lack properties such as error bounds or preservation of stability, passivity, or phase information. A comparison of the numerical properties of SVD-based methods (as BT) and Krylov subspace methods can be found in [1,5,6,7].

The Lyapunov equations are solved in our method via the matrix sign function, which yields a computational cost for the global model reduction procedure of $\mathcal{O}(n^3)$ flops (floating-point arithmetic operations). This calls for the application of high performance computing in the reduction of models already with n in the order of thousands.

Recent work on the implementation of BLAS and the major factorization routines for the solution of linear systems [8,9,10,11] has demonstrated the potential of graphics processors (GPUs) to yield high performance on dense linear algebra operations which can be cast in terms of matrix-matrix products. In [12] we built upon these works to deal with the solution of the standard Lyapunov equation on a GPU. Here, we further extend this work by tackling the different stages in SVD-based methods for model reduction of generalized linear systems, namely, the solution of the coupled generalized Lyapunov equations, the computation of the SVD, and auxiliary computations. The target architecture is a hybrid platform consisting of a general-purpose multicore processor and a GPU. We exploit these two resources by designing a hybrid numerical algorithm for model reduction that performs fine-grained computations on the CPU while off-loading computationally intensive operations to the GPU. We also overlap computations on both architectures in order to improve the performance.

The rest of the paper is structured as follows. In Section 2 we briefly review the BT method for model reduction, including the sign function-based Lyapunov

solver, and the remaining stages of the method. There we also describe the approach to computing all these operations on the hybrid platform. In Section 3 we present experimental results that illustrate the accuracy and parallelism attained by the numerical algorithms on a platform consisting of two Intel QuadCore processors connected to an NVIDIA Tesla C1060 GPU via a PCI-e bus. Finally, in Section 4 we provide a few concluding remarks.

2 SVD-Based Methods for Model Reduction

In the following, we assume the matrix pair (A, E) to be stable (i.e., all its generalized eigenvalues are in the open left complex plane), which is often the case in model reduction applications.

BT model reduction [13,14,15,16] belongs to the family of absolute error methods, which aim at minimizing

$$\|G - \hat{G}\|_\infty = \sup_{\omega \in \mathbb{R}} \sigma_{\max}(G(j\omega) - \hat{G}(j\omega)), \tag{3}$$

where $j := \sqrt{-1}$ and $\sigma_{\max}(M)$ stands for the largest singular value of a matrix M.

Model reduction via BT methods employs information about the controllability Gramian W_c and the observability Gramian W_o of the system (1), obtained by the solutions of the coupled generalized Lyapunov matrix equations

$$AW_cE^T + EW_cA^T + BB^T = 0, \tag{4}$$
$$A^T\tilde{W}_oE + E^T\tilde{W}_oA + C^TC = 0, \tag{5}$$

with $W_o = E^T\tilde{W}_oE$. The stability of (A, E) implies that W_c and W_o are both symmetric positive semidefinite. Therefore, the solutions of the Gramians can be factored as $W_c = S^TS$ and $W_o = R^TR$. We follow the standard convention and call S and R Cholesky factors of W_c and W_o, respectively, despite the fact that they are not square and upper triangular as required from a Cholesky factor in the strict sense.

Consider now the singular value decomposition (SVD) of the product

$$SR^T = U\Sigma V^T = [\, U_1 \; U_2 \,] \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} [\, V_1 \; V_2 \,]^T, \tag{6}$$

where U and V are orthogonal matrices, and $\Sigma = \mathrm{diag}\,(\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a diagonal matrix containing the singular values of SR^T, also known as the *Hankel singular values* (HSV) of the system. Given a partitioning of Σ into $\Sigma_1 \in \mathbb{R}^{r\times r}$ and $\Sigma_2 \in \mathbb{R}^{(n-r)\times(n-r)}$, and a conformal partitioning of U and V in (6), the *square-root* (SR) version of BT determines a reduced-order model of order r as

$$\begin{array}{l} \hat{E} = T_lET_r,\ \hat{A} = T_lAT_r, \\ \hat{B} = T_lB, \quad \hat{C} = CT_r, \quad \hat{D} = D, \end{array} \tag{7}$$

with

$$T_l = \Sigma_1^{-1/2} V_1^T R E^{-1} \quad \text{and} \quad T_r = S^T U_1 \Sigma_1^{-1/2}. \tag{8}$$

The state-space dimension r of the reduced-order model can be chosen adaptively as this method provides a realization $\hat{G}$ satisfying the error bound

$$\|G - \hat{G}\|_\infty \leq 2 \sum_{j=r+1}^{n} \sigma_j. \tag{9}$$

In the following subsection we revisit the sign function-based generalized Lyapunov solver introduced in [17]. The solver yields low-rank approximations to the Cholesky or full-rank factors of the solution matrices which can reliably substitute S and R in the computations in (6) and (8).

2.1 The sign function method

The matrix sign function was introduced in [18] as an efficient tool to solve stable (standard) Lyapunov equations. The following variant of the Newton iteration for the matrix sign function [17] can be used for the solution of the generalized Lyapunov equations (4)–(5):

Algorithm `CGCLNC`**:**

$A_0 \leftarrow A$, $\tilde{S}_0 \leftarrow B^T$, $\tilde{R}_0 \leftarrow C$
$k \leftarrow 0$
`repeat`
  $A_{k+1} \leftarrow \frac{1}{\sqrt{2}} \left(A_k/c_k + c_k (E A_k^{-1}) E \right)$
  Compute the rank-revealing QR (RRQR) decomposition
    $\frac{1}{\sqrt{2c_k}} \left[\tilde{S}_k, \;\; c_k \tilde{S}_k (E A_k^{-1})^T \right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$
  $\tilde{S}_{k+1} \leftarrow U_s \Pi_s$
  Compute the RRQR decomposition
    $\frac{1}{\sqrt{2c_k}} \left[\tilde{R}_k, \;\; c_k (\tilde{R}_k A_k^{-1}) E \right] = Q_r \begin{bmatrix} U_r \\ 0 \end{bmatrix} \Pi_r$
  $\tilde{R}_{k+1} \leftarrow U_r \Pi_r$
  $k \leftarrow k + 1$
`until` $\|A_k - E\|_1 < \tau \|A_k\|_1$

On convergence, after j iterations, $\tilde{S} = \frac{1}{\sqrt{2}} \tilde{S}_j E^{-T}$ and $\tilde{R} = \frac{1}{\sqrt{2}} \tilde{R}_j E^{-1}$ are of dimensions $\tilde{k}_o \times n$ and $\tilde{k}_c \times n$ and represent full (row-)rank approximations of S and R respectively, so that $W_c = S^T S \approx \tilde{S}^T \tilde{S}$ and $W_o = R^T R \approx \tilde{R}^T \tilde{R}$.

The Newton iteration for the sign function usually enjoys quick convergence, which is ultimately quadratic. Initial convergence can be accelerated using several techniques. In our case, we employ a scaling defined by the parameter

$$c_k = \sqrt{\|A\|_\infty / \|E A_k^{-1} E\|_\infty}.$$

In the convergence test, τ is a tolerance threshold for the iteration that is usually set as a function of the problem dimension and the machine precision ε. In particular, to avoid stagnation in the iteration, we set $\tau = n \cdot \sqrt{\varepsilon}$ and perform one or two additional iteration steps after the stopping criterion is satisfied. Due to the quadratic convergence of the Newton iteration, this is usually enough to reach the attainable accuracy. The RRQR decomposition can be obtained by means of the traditional QR factorization with column pivoting [19] plus a reliable rank estimator.

Each iteration of algorithm `CGCLNC` requires the following operations: the LU decomposition of A_k ($\frac{2}{3}n^3$ flops), followed by the system solve EA_k^{-1} and the matrix product $(EA_k^{-1})E$ ($2n^3 + 2n^3$ flops). If s_k and r_k denote the number of columns of $\hat{S}_k$ and $\hat{R}_k$, then one matrix product is required to construct $\tilde{S}_k(EA_k^{-1})^T$ ($2n^2 s_k$ flops), a system solve with r_k right-hand sides to obtain $\tilde{R}_k A_k^{-1}$ ($2n^2 r_k$ flops), and an $n \times n \times r_k$ matrix product to build $(\tilde{R}_k A_k^{-1})E$ ($2n^2 r_k$ flops). Finally, two QR factorizations with column pivoting complete the major computations of the algorithm ($2n(s_k^2 + r_k^2) - \frac{2}{3}(s_k^3 + r_k^3)$ flops). (The latter flop count assumes that S_k and R_k are full rank, so the actual cost is smaller than this.) Other minor operations, as norms, scalings, etc., contribute with negligible computational costs.

2.2 Hybrid implementation of the Lyapunov solver

The objective of the hybrid implementation is to reduce the computational time executing each operation on the most convenient architecture while, whenever possible, overlapping the execution of operations on both architectures. On the other hand, a careful scheduling of operations is necessary to minimize the communication overhead, amortizing the cost of transferring the data between the memory spaces of the GPU and the CPU.

The hybrid algorithm proceeds as follows. At the beginning of each iteration, the CPU transfers matrix A_k to the GPU. Then, the CPU and the GPU cooperate in the LU factorization of the matrix A_k. The solution of the EA_k^{-1} system is also obtained on the GPU while the CPU solves the $\tilde{R}_k A_k^{-1}$ system. Then, the computation of the matrix product $(EA_k^{-1})E$ proceeds on the GPU while the CPU executes some minor operations.

The rest of the operations are performed on the CPU since they require a minor computational effort.

The use of both architectures requires some data transfers. To control and minimize the communication overhead, data transfers are only scheduled if there is an important gain associated with them. Specifically, the data transfers needed at each iteration are:

1. Send A_k from the CPU to the GPU to compute its LU decomposition.
2. Send the factors resulting from the LU decomposition of A_k from the GPU to the CPU.
3. Send the solution of EA_k^{-1} from the GPU to the CPU.
4. Send the result of $(EA_k^{-1})E$ from the GPU to the CPU.

Besides these data transfers, there are some minor communications being performed in the algorithm, in particular, by the LU decomposition kernel.

In summary, the most remarkable strengths of this implementation are:

- The use of a hybrid kernel for the LU decomposition. In this kernel the GPU and the CPU cooperate for computing the decomposition [10].
- The new routine for the solution of triangular systems on the GPU. This version outperforms notoriously the CUBLAS implementation (it is approximately a 30% and 70% faster for the examples `STEEL_I` and `FLOW_METER` considered in Section 3, respectively) and yields a significant acceleration of one of the most time-consuming stages in the model reduction procedure.
- The use of two levels of parallelism. At the inner level, operations are performed using multi-threaded implementations of BLAS. At the outer level, different operations are executed concurrently in the two available resources: CPU and GPU.
- The reduced overhead introduced by communications: only transfers that are amortized are performed, so that it will be unlikely that a communication produces a loss of efficiency.

2.3 Remaining stages in model reduction BT

Once the Cholesky factors $\tilde{S}$ and $\tilde{R}$ have been computed, the remaining operations to obtain the reduced order model comprise a matrix product of moderate dimension ($\tilde{S}^T\tilde{R} \approx SR^T$); the SVD of the result, see (6); and a few more matrix-matrix operations and a system solve, see (7)–(8). All these computations require a moderate number of flops and, therefore, are performed on the CPU.

3 Numerical Experiments

In this section we evaluate the numerical accuracy and parallel performance of the BT model reduction method. The target platform consists of two Intel Xeon QuadCore E5410 processors at 2.33GHz, connected to an Nvidia Tesla C1060 via a PCI-e bus (see Table 1 for more details). We employed the multi-threaded implementation of BLAS in MKL (version 10.2) for the general-purpose processor and Nvidia CUBLAS (version 2.1) for the GPU. We set `OMP_NUM_THREADS=8` so that one thread is employed per core in the parallel execution of the MKL routines in the Intel Xeon QuadCore processors.

In the following experiments, we evaluate the performance of the method using two model reduction problems from the Oberwolfach benchmark collection at the University of Freiburg[1]:

- `STEEL_I`: This model arises in a manufacturing method for steel profiles. The goal is to design a control that yields moderate temperature gradients when the rail is cooled down. The mathematical model corresponds to the

[1] `http://www.imtek.de/simulation/benchmark/`.

Processors	#cores	Freq. (GHz)	L2 (MB)	Memory (GB)
INTEL Xeon	8	2.3	12	8
NVIDIA Tesla	240	1.3	–	4

Table 1. Hardware employed in the experiments.

boundary control for a 2-D heat equation. A finite element discretization, followed by adaptive refinement of the mesh results in the example in this benchmark. The dimensions of this problem are $n = 5,177$, $m = 7$, $p = 6$.

– `FLOW_METER`: This 2-D model of an anemometer-like structure mainly consists of a tube and a small heat source. The model is given by a spatially semi-discretized instationary convection-diffusion equation with Dirichlet boundary conditions and a parabolic inflow profile. Both, Dirichlet boundary and initial conditions are set to the reference temperature of 300 K. The dimensions of this problem are $n = 9,669$, $m = 1$, $p = 5$.

Table 2 shows the results obtained with our hybrid CPU-GPU algorithm for the solution of the coupled generalized Lyapunov equations associated with these systems. Columns 2, 3, 4 and 5 of the table show the time (in seconds) for the LU factorization of A_k, the solution of the four triangular systems in the computations EA_k^{-1} and $\tilde{R}_k A_k^{-1}$, the matrix product $(EA_k^{-1})E$, and the updates of the factors $\tilde{S}$ and $\tilde{R}$, respectively (including the time for all the data transfers associated to each one of the operations). Columns 6, 7, and 8 show the global time per iteration of the hybrid implementation, the time per iteration for the same algorithm implemented on the multicore CPU, and the convergence criterion.

Most of the iteration time is spent in the computation of the LU decomposition (column 2), the solution of the four triangular systems (column 3) and the matrix-matrix product (column 4). Those are the operations which, in part or completely, are performed on the GPU.

The number of columns of the factors $\tilde{S}$ and $\tilde{R}$ is doubled at each iteration and, in consequence, the cost associated to the update of the factors increases with the iteration count. To keep the number of columns in the factors under control, an RRQR factorization is computed at each step [19]. This approach yields important gains when the number of iterations that are required for convergence is large enough to increment notoriously the size of the factors, as is the case for the two problems considered in this section. The increment in the number of columns of $\tilde{S}$ and $\tilde{R}$ results in an increment of the time required for their update (column 5). This time, after some iterations, becomes nearly half of the total iteration time, due to the fact that this is mostly a BLAS-2 based computation performed on the CPU. Executing these operations on the GPU, though possible, would require some extra CPU-GPU communications and would slow down the execution of the initial iterations.

#Iter k	Time $PA_k = LU$	Time EA_k^{-1}, $\tilde{R}_k A_k^{-1}$	Time $(EA_k^{-1})E$	Time $\tilde{S}_k(EA_k^{-1})$, $\tilde{R}_k(A_k^{-1}E)$, compress	Time iteration (Hybrid)	Time iteration (CPU)	Conv. criterion $\frac{\|A_k+E\|_F}{\|E\|_F}$
STEEL_I							
1	0.698	1.041	0.807	0.121	2.958	5.337	2.732e+02
2	0.544	1.023	0.788	0.047	2.618	5.286	2.064e+01
3	0.544	1.023	0.788	0.079	2.650	5.354	3.698e+00
4	0.544	1.023	0.788	0.159	2.732	5.465	1.140e+00
5	0.543	1.023	0.789	0.381	2.955	5.638	3.644e−01
6	0.545	1.023	0.788	0.909	3.486	6.219	7.936e−02
7	0.546	1.022	0.789	1.366	3.946	6.553	8.546e−03
8	0.543	1.023	0.788	1.866	4.442	6.909	5.706e−04
9	0.544	1.184	0.788	2.093	4.670	7.105	1.257e−05
10	0.546	1.209	0.788	2.185	4.767	7.250	7.319e−07
ACCUMULATED TIME					35.224	61.156	
FLOW_METER							
1	3.380	7.741	5.183	0.289	17.359	31.516	6.884e+01
2	2.906	7.673	5.116	0.109	16.512	31.580	6.758e+00
3	2.918	7.673	5.116	0.137	16.553	31.725	1.585e+00
4	2.888	7.673	5.116	0.202	16.592	31.970	5.010e−01
5	3.007	7.673	5.115	0.359	16.871	32.126	1.580e−01
6	2.893	7.674	5.116	0.702	17.099	32.329	5.044e−02
7	2.886	7.673	5.116	0.971	17.365	32.525	1.241e−02
8	2.890	7.674	5.116	1.066	17.462	32.842	1.702e−03
9	2.893	7.673	5.117	1.191	17.591	32.896	1.156e−04
10	2.891	7.673	5.115	1.236	16.634	32.997	1.396e−06
11	2.891	7.673	5.116	1.248	17.994	32.881	2.389e−07
ACCUMULATED TIME					188.032	355.387	

Table 2. Performance of the hybrid CPU+GPU implementation of the Newton iteration for the solution of the Lyapunov equation with factored right-hand side.

Compared with the execution of the same algorithm on a CPU, the use of the GPU yields an important reduction of the execution times on the most computationally expensive operations which carries over to the global execution time per iteration (the LU factorization, the solution of triangular systems and the matrix-matrix product). Furthermore, while some computations are off-loaded to the GPU, others are performed concurrently on the CPU. This second level of parallelism further reduces the total execution time.

4 Concluding Remarks

We have presented a new parallel algorithm for model reduction of large linear systems on a hybrid CPU-GPU platform. Our algorithm exploits the capabilities of both architectures, the multi-core CPU and the many-core GPU, obtaining

a high performance implementation of a BT model reduction technique. We use two levels of parallelism: at the inner level, multi-thread implementations of the BLAS library (MKL and CUBLAS) compute the most time-consuming linear algebra kernels. At the outer level, operations proceed concurrently in both architectures.

Results show that model reduction of large-scale linear systems can be tackled with this kind of platforms in a reasonable computational time.

Future research resulting from this experience will include:

- The use of multiple GPUs to further reduce the computational time and increase the dimension of the affordable problems.
- Use double precision arithmetic. Performance of current GPUs in double precision is considerably lower than single precision, but the new generation of GPUs will drastically reduce this difference. As an alternative, we will investigate the use of iterative refinement which given a single precision solution, obtains the double precision solution at a reduced cost.

References

1. A. Antoulas, *Approximation of Large-Scale Dynamical Systems.* Philadelphia, PA: SIAM Publications, 2005.
2. P. Benner, E. Quintana-Ortí, and G. Quintana-Ortí, "State-space truncation methods for parallel model reduction of large-scale systems," *Parallel Comput.*, vol. 29, pp. 1701–1722, 2003.
3. T. Penzl, "Algorithms for model reduction of large dynamical systems," *Linear Algebra and its Applications*, vol. 415, no. 2-3, pp. 322 – 343, 2006.
4. R. Freund, "Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation," in *Applied and Computational Control, Signals, and Circuits*, B. Datta, Ed. Boston, MA: Birkhäuser, 1999, vol. 1, ch. 9, pp. 435–498.
5. P. Benner, "Numerical linear algebra for model reduction in control and simulation," *GAMM-Mitteilungen*, vol. 29, no. 2, pp. 275–296, 2006.
6. P. Benner, V. Mehrmann, and D. Sorensen, Eds., *Dimension Reduction of Large-Scale Systems*, ser. Lecture Notes in Computational Science and Engineering, vol. 45. Springer-Verlag, 2005.
7. W. Schilders, H. van der Vorst, and J. Rommes, Eds., *Model Order Reduction: Theory, Research Aspects and Applications*, ser. Mathematics in Industry, vol. 13. Springer-Verlag, 2008.
8. V. Volkov and J. Demmel, "LU, QR and Cholesky factorizations using vector capabilities of GPUs," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-49, May 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-49.html
9. P. Bientinesi, F. D. Igual, D. Kressner, and E. S. Quintana-Ortí, "Reduction to condensed forms for symmetric eigenvalue problems on multi-core architectures," in *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics – PPAM'09*, ser. Lecture Notes in Computer Science. Springer, to appear.

10. S. Barrachina, M. Castillo, F. D. Igual, R. Mayo, E. S. Quintana-Ortí, and G. Quintana-Ortí, "Exploiting the capabilities of modern GPUs for dense matrix computations," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 2457–2477, 2009.
11. H. Ltaif, S. Tomov, R. Nath, P. Du, and J. Dongarra, "A scalable high performance cholesky factorization for multicore with GPU accelerators," University of Tennessee, LAPACK Working Note 223, 2009.
12. P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, "Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function," in *Euro-Par 2009, Parallel Processing - Workshops*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2009, vol. 6043, pp. 132–139.
13. B. Moore, "Principal component analysis in linear systems: Controllability, observability, and model reduction," *IEEE Trans. Automat. Control*, vol. AC-26, pp. 17–32, 1981.
14. M. Safonov and R. Chiang, "A Schur method for balanced-truncation model reduction," *IEEE Trans. Automat. Control*, vol. AC–34, pp. 729–733, 1989.
15. M. Tombs and I. Postlethwaite, "Truncated balanced realization of a stable non-minimal state-space system," *Internat. J. Control*, vol. 46, no. 4, pp. 1319–1330, 1987.
16. A. Varga, "Efficient minimal realization procedure based on balancing," in *Prepr. of the IMACS Symp. on Modelling and Control of Technological Systems*, vol. 2, 1991, pp. 42–47.
17. P. Benner, E. Quintana-Ortí, and G. Quintana-Ortí, "Solving linear-quadratic optimal control problems on parallel computers," *Optimization Methods Software*, vol. 23, no. 6, pp. 879–909, 2008.
18. J. Roberts, "Linear model reduction and solution of the algebraic Riccati equation by use of the sign function," *Internat. J. Control*, vol. 32, pp. 677–687, 1980, (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).
19. G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed. Baltimore: Johns Hopkins University Press, 1996.